Implementation

Problem 1

The implementation for this problem was quite simple. First, read the data using Hatchet's **from_caliper( )** function. Then, I used pandas operations to remove the columns I didn't need, that is only keep time and name. Then I used **df.sort_values( )** to sort the rows based on the 'time' column in descending order. Then I loop over the rows in df printing each row's name and time as required and decrement **n** each time I print, when n = 0 I stop printing. I probably should have used a while loop instead of a for loop with a conditional break, but it is what it is.

Problem 2

This one was again, pretty simple to implement. The only issue was the definition for how the Xth row from the top was wonky and needlessly convoluted. I had to clarify on Piazza because I wasn't sure if 1st from the top meant the 0th row or the 1st row. To implement this problem I first read the data in using **from_caliper( ).** Then I used the **load_imbalance( )** function on the time column as required. Then I sorted the rows in descending order according to the time.imbalance column using **df.sort_values( )**. Then I use **iloc[x-1]** to grab the desired row, once I have the desired row I can print **row['time.ranks']** to get the desired output.

Problem 3

Once again, this was easy the only issue that came up was a warning due to hatchet's internal implementation which confused me. I first read in both datasets using **from_caliper( )**. Then I call **drop_index_levels**( ) on each of them. Then I create a new variable that holds (g64 -

g32) which is the difference between the two graph frames. Then I grab the important columns in the data frame and store them in the df variable. Then I sort the rows according to the time column in descending order using **df.sort_values( )**. Then I loop over the rows in df printing each row's name and time as required and decrement **n** each time I print, when n = 0 I stop printing.

Hatchet Function Highlight

I will be honest, this write-up as a whole feels pretty pointless, but this question seems extra pointless. There are only really three hatchet functions I used **from_caliper( )**, **load_imbalance( )**, and **drop_index_levels( )**. There is no way to extract any perceivable value out of using **load_imbalance( )** or **drop_index_levels( )** because I only used it due to the assignment instructions and not to make my life any easier. I suppose the most useful function is **from_caliper( )** because it unlocks the data frame which was the basis for most of this assignment.

If we widen the scope to the most useful stuff I used in the entire project as a whole, then I would say having access to Hatchet to convert the data into a pandas data frame is the most useful thing, then it's the pandas operations I can execute on that data frame. Access to that pandas dataframe allows you to do a lot of different operations, whereas the other functions are only useful in specialized cases.